042390.P11391 *Patent*

UNITED STATES PATENT APPLICATION

FOR

INTELLIGENTLY DETERMINING WHICH TRAFFIC STREAMS TO OFFLOAD EFFICIENTLY

INVENTORS:

AVRAHAM MUALEM
LINDEN MINNICK

PREPARED BY:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN, LLP
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CA 90025-1026

(503) 684-6200

EXPRESS MAIL NO. EL625195539US

INTELLIGENTLY DETERMINING WHICH TRAFFIC STREAMS TO OFFLOAD EFFICIENTLY

FIELD OF THE INVENTION

[0001] The present invention relates to performing cryptography operations on data streams. More particularly, the present invention relates to IP security offload.

BACKGROUND OF THE INVENTION

[0002] IP Security (IPsec) standard, IP Security Internet Engineering Task Force (IETF) Request for Comments (RFC) 2401, published November 1998, is one method of protecting both the confidentiality and integrity of data transferred on a network. Because IPsec provides a way to encrypt and decrypt data below the Transmission Control Protocol (TCP)/User Datagram Protocol (UDP) layer, the protection is transparent to applications that transfer data. Thus, a system may utilize IPsec without requiring changes at the application level. However, the algorithms used for cryptography (crypto) operations (e.g., encryption, decryption, authentication) on the data for IPsec require many processor cycles to execute. The processor cycles spent on crypto operations decrease the cycles available to applications and other parts of the protocol stack. This in turn decreases throughput in the system.

[0003] One solution to this problem is to offload the crypto operations to an external piece of hardware, such as a Network Interface Card (NIC). One way to offload the crypto operations is by encrypting data immediately before transmitting a packet and decrypting data directly from the wire before the packet is transferred via Direct Memory Access (DMA) to host memory. A Security Association (SA) is a data structure of cryptography information that contains all of the information necessary to perform crypto operations on a packet. The device that interfaces the system to the network, for example a NIC, detects which SA is needed to process the packet and

performs crypto operations on the packet directly from the wire. The process for performing crypto operations on ingress data before it is transferred to host memory is called "Inline Receive."

[0004] Another solution is to offload using the "Secondary Use" model. This model uses an out-of-band acceleration method to decrypt receive packets. In this scenario, all received packets are DMA transferred to host memory. The network interface driver then parses each received packet to match it with its corresponding Security Association (SA). Assuming that the crypto accelerator is on the NIC, the driver then instructs the NIC to transfer the packet across the bus, perform the crypto operation on the packet, and then transfer the packet back to host memory. Secondary Use results in inefficient use of the available bandwidth on the system bus because the packet is transferred across the bus three times. Secondary Use also creates additional latency, which can degrade the throughput of protocols sensitive to the round trip time of packets (e.g., TCP). Furthermore, performing the extra transfers across the bus often requires the use of an additional interrupt, causing the system to do more work, and increasing CPU utilization.

[0005] The network interface driver must determine which traffic streams will be handled using Inline Receive, and which traffic streams will be handled using either Secondary Use or software processing. A traditional approach is to add SAs to the Inline Receive cache on a first come, first served basis. Under the traditional approach, packets associated with SAs in the cache will be handled using Inline Receive. Once there are no available entries in the cache, packets for non-cached SAs must be handled using either Secondary Use or software processing.

[0006] From a performance perspective (both processor utilization and throughput), Inline Receive is a more efficient solution than Secondary Use. On the other hand, Inline Receive is expensive to implement in hardware because the SAs necessary for the crypto operations must be

stored in an SA cache on the NIC. Because of this, the number of connections that can use Inline Receive is often limited. Because it is common for the number of open connections to exceed the size of the Inline Receive cache, the other connections must use the Secondary Use model in order to offload secure traffic.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings in which like reference numerals refer to similar elements.

Figure 1 is one embodiment of a block diagram of an electronic system.

Figure 2 is one embodiment of a block diagram of a network interface system.

Figure 3 is one embodiment of a block diagram of a driver agent.

Figure 4 is one embodiment of a flow diagram for adding a security association to a driver agent.

Figure 5 is one embodiment of a flow diagram for handling a packet received by a driver agent.

Figure 6 is one embodiment of a flow diagram for replacing security association entries in a cache.

Figure 7 is one embodiment of a flow diagram for decreasing the value of a security association metric.

DETAILED DESCRIPTION

[0007] Methods and apparatuses for dynamically mapping traffic streams to one of multiple methods to perform cryptography operations based on network traffic analysis are described. In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the present invention can be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid obscuring the present invention.

[0008] Reference in the specification to "one embodiment" or "an embodiment" means that a particular feature, structure or characteristic described in connection with the embodiment is included in at least one embodiment of the present invention. Thus, the appearances of the phrase "in one embodiment" appearing in various places throughout the specification are not necessarily all referring to the same embodiment. Likewise, the appearances of the phrase "in another embodiment," or "in an alternate embodiment" appearing in various places throughout the specification are not all necessarily all referring to the same embodiment.

[0009] Briefly, a technique for associating a metric value with a Security Association (SA) for each network traffic stream coupled through a network interface to an electronic system is described. The metric is used to determine which of multiple methods to perform cryptography (crypto) operations (e.g., encryption, decryption, authentication) should be used to handle which streams.

[0010] The crypto operations can be offloaded to improve system performance. Offloading is a process whereby the system processor does not perform the crypto operations. One form of offloading is "Inline" Operation. With Inline Operation, hardware external to the system

processor, for example a processor or co-processor on a Network Interface Card (NIC), performs the crypto operations directly off the wire. For example, a packet entering the system would be operated on before the packet was transferred to host memory. Another method is "Secondary Use." With Secondary Use, a driver agent, for example a software network interface driver, uses hardware external to the system processor, for example a processor or co-processor on a NIC, to perform the operations. For example, a packet entering the system would be transferred to host memory, and the network interface driver would instruct the NIC hardware to process the packet. Inline Receive is a more efficient method of offloading than Secondary Use in terms of system performance.

[0011] In one embodiment, an electronic system uses a predetermined policy to decide to handle one network traffic stream with Inline Operation and another network traffic stream with the Secondary Use model. For example, a network traffic stream with heavy traffic will be handled with Inline Receive, and a network traffic stream with lesser traffic will be handled with the Secondary Use model. By determining which of multiple methods to perform crypto operations to use with which traffic streams, system performance is increased.

[0012] Figure 1 is one embodiment of an electronic system. Electronic system 100 may be, for example, a computer, a Personal Digital Assistant (PDA), a set top box, or any other electronic system. System 100 includes bus 101 or other communication device to communicate information, and processor 102 coupled with bus 101 to process information and to execute instructions. System 100 further includes memory 103, coupled to bus 101 to store information and instructions to be executed by processor 102. Memory 103 may also be used to store temporary variables or other intermediate information during execution of instructions by

processor 102. Memory 103 may include random access memory (RAM), read-only memory (ROM), flash, or other static or dynamic storage media.

[0013] User interfaces 104 are coupled to bus 101 too allow interaction with a user. Mass storage 105 can be coupled to system 100 to provide instructions to memory 103. Mass storage 105 can be, for example, a magnetic disk or optical disc and its corresponding drive, a memory card, or another device capable of storing machine-readable instructions. Network interfaces 106 can be coupled to bus 101 to enable system 100 to communicate with other electronic systems via a network. Driver agent 107 may be coupled to system 100 to perform driver features in hardware. Driver agent 107 may be an Application Specific Integrated Circuit (ASIC), a special function controller or processor, a Field Programmable Gate Array (FPGA), or other hardware device to perform the functions of a driver. Driver agent 107 is not a necessary part of system 100.

[0014] Electronic system 100 can improve system performance by determining which components of system 100 should handle traffic streams coupled to bus 101 by network interfaces 106. According to one embodiment, a driver agent determines which traffic streams should be mapped to more efficient techniques of performing crypto operations based on the value of a corresponding SA metric. For example, the driver agent can use a predetermined policy to compare metric values and use the most efficient technique to handle traffic stream corresponding to the highest metric value. The driver agent can be driver agent 107 or a software driver agent incorporated from a series of machine-readable instructions stored within memory 103.

[0015] Instructions can be provided to memory 103 from a storage device, such as magnetic disk, CD-ROM, DVD, via a remote connection (e.g., over a network), etc. In alternative

embodiments, hard-wired circuitry can be used in place of or in combination with software instructions to enable system 100 to practice the present invention. Thus, the electronic system depicted above is not limited to any specific combination of hardware circuitry and software structure.

[0016] Instructions can be provided to memory 103 from a form of machine-accessible medium. A machine-accessible medium includes any mechanism that provides (i.e., stores and/or transmits) information in a form readable by a machine (e.g., a computer). For example, a machine-accessible medium includes read only memory (ROM); random access memory (RAM); magnetic disk storage media; optical storage media; flash memory devices; electrical, optical, acoustical or other form of propagated signals (e.g., carrier waves, infrared signals, digital signals); etc.

[0017] **Figure 2** is one embodiment of a block diagram of a network interface system coupled to a network. In one embodiment, Network Interface (NI) 210 is a communication interface that enables an electronic system to communicate with other electronic systems coupled to network 220. For example, NI 210 can be a Network Interface Card (NIC). In one embodiment, traffic streams are received from network 220 into buffer 211 on NI 210. NI processor 215 determines whether NI 210 has the SA information in cache 212 necessary to perform crypto operations on the packet. If the SA information is in cache 212, crypto services 216 handles the packet. If the SA information is not in cache 212, the packet is transferred across bus 101 to memory 103.

[0018] Memory 103 contains operating system (OS) 231 which controls the flow of instructions to processor 102. In one embodiment, OS 231 is the highest layer of control of the electronic system. Driver agent 233 is a lower layer of system control. In one embodiment, SAs are delivered to driver agent 233 by OS 231. In another embodiment, applications 232 can contain

agents of a higher layer of control than driver agent 233 and deliver SAs to driver agent 233. Applications 232 can also contain other programs (e.g., word processor(s); electronic mail (e-mail) programs). Memory 103 also contains SA table 234 that is a data structure of SAs. In one embodiment, driver agent 233 accesses SA table 234 to determine how to map traffic streams to one of multiple components that perform crypto operations. In one embodiment, crypto services 216 and driver agent 233 are components that perform crypto operations.

[0019] Processor 102 performs operations based on instructions received from memory 103. For example, sets of instructions can control timers for use by functions that must be performed on a periodic basis. In one embodiment, driver agent 233 responds to the use of a timer to periodically determine whether SA entries of cache 212 should be replaced by non-cached SAs.

[0020] **Figure 3** is one embodiment of a block diagram of a driver agent. Control logic 310 directs the flow of operation of driver agent 300. In one embodiment, control logic 310 is a series of software instructions to perform logic operations. In another embodiment, control logic 310 can be implemented by hardware control logic, or a combination of hardware-based control logic and software instructions.

[0021] System interfaces 340 couples driver agent 300 to an electronic system. For example, in one embodiment, driver agent 300 can be part of a computer system and system interfaces 340 couples driver agent 300 to the computer system via a system bus. Thus, control logic 310 can receive a series of instructions from application software external to driver agent 300. Driver agent 300 is not limited to being local to an electronic system. For example, system interfaces 340 may couple driver agent 300 to an electronic system through a network.

[0022] In one embodiment, driver agent 300 may contain applications 320. Thus, applications 320 can supplement external instructions to control logic 310. Applications 320 are not necessary to the function of driver agent 300.

[0023] In one embodiment, mapping feature 351 dynamically maps traffic streams to one of multiple components that perform crypto operations. For example, mapping feature 351 can enable driver agent 300 to direct a NIC to handle a packet received into host memory by the Secondary Use model. In one embodiment, SA metric operation feature 352 enables driver agent 300 to operate on the value of SA metrics. For example, driver agent 300 can initialize, increment, add value to, or decrease the SA metrics.

[0024] In one embodiment, cache replacement feature 353 enables driver agent 300 to intelligently replace SA entries in a cache. For example, cache replacement feature 353 can enable driver agent 300 to determine whether to replace entries in an SA cache on a NIC based on SA metric values. Mapping feature 351, SA metric operation feature 352, and cache replacement feature 353 can exist independently of and be external to driver agent 300. For example, upper layers of an electronic system may perform the same or similar functions as mapping feature 351, SA metric operation feature 352, and cache replacement feature 353. Likewise, driver engine 350 may exist as a more complex or less complex embodiment, containing some, all, or additional features to those represented in Figure 3.

[0025] **Figure 4** is one embodiment of a flow diagram for adding an SA to a driver agent. In one embodiment, the upper system layers add an SA to a network interface driver, 410. The upper layers can be, for example, an OS or another application that performs control functions. A network interface driver is one embodiment of a driver agent. In one embodiment, the SA is

added to the network interface driver by the upper layers as one step in the execution of a set of instructions for offloading a received packet.

[0026] The SA added in 410 is associated with an SA metric and initialized to a predetermined value **Y**, 420. In one embodiment, **Y** is a value greater than zero that is selected based on experimentation, for example, 10,000.

[0027] The driver agent determines whether there is a cache entry available, 430. If there is a cache entry available, the SA is added to the cache and a value **X** is added to the SA metric, 440. In one embodiment, the value of **X** is predetermined based on experimentation, for example, 10,000. Adding **X** to the SA metric according to the present invention will give advantage to the SA added to the cache so that it is likely remain in the cache. If there is no cache entry available, the SA is not added to the cache, 450. In one embodiment, SAs not cached remain in an SA table in system memory.

[0028] **Figure 5** is one embodiment of a flow diagram for handling a packet received by a driver agent. In one embodiment, a packet associated with a traffic stream is received by a driver agent and the SA information is extracted, 510. For example, a packet can be received by a NIC and DMA transferred to host memory, and a network interface driver can parse out its SA information. The SA information is extracted to determine which SA is associated with the packet, and contains all the information necessary to perform crypto operations on the packet.

[0029] In one embodiment, the metric value of the associated SA is incremented, 520. In one embodiment, for example, a driver agent will perform a numerical operation on the SA metric. Crypto operations are performed on the packet, 530. In one embodiment, for example, the driver agent will direct a NIC to perform crypto operations on the packet by Secondary Use.

Alternatively, the packet may be handled using software processing with any crypto operation known in the art.

[0030] **Figure 6** is one embodiment of a flow diagram for replacement of SA entries in a cache. A timer **T1** expires indicating a timeout condition, 610. A timer can be any form of counter, timer, capture and compare, or other device or method used to indicate to an electronic system process when a time-dependent operation or series of operations should be performed. Detection of the timeout condition may be performed by either software or hardware interrupt, by timer functions provided by the operating system, or by polling. In one embodiment, the timeout period for **T1** is determined by experimentation to minimize SA replacement cost, for example, one second. Thus, cache replacement is performed based on time, rather than caused by a cache miss.

[0031] In one embodiment, the expiration of timer **T1** indicates to a system that a predetermined policy for cache replacement should be performed. For example, a system processor can have a timer that causes an interrupt on overflow to indicate when to execute a command to a driver agent to perform a function for determining whether entries in an SA cache on a NIC should be replaced. For example, a cache replacement feature can enable a driver agent to perform a cost-based analysis to determine when a cached SA should be replaced by a non-cached SA.

[0032] It is determined whether there are any non-cached SAs, 620. In one embodiment, SAs not in the cache will exist in memory within the system. For example, SAs not in an SA cache on a NIC can be stored in an SA table in system memory. If all SAs are cached, no entries will be replaced, 690. If there are non-cached SAs, the list of cached SAs is searched for the best candidate (**B**) for eviction, 630. In one embodiment, **B** is the SA with the lowest SA metric value. For example, the metric associated with cached SA **B** may be periodically decreased by a

driver agent, and the traffic stream associated with **B** may concurrently have a long period of disuse, resulting in a low SA metric value.

[0033] The list of non-cached SAs is searched for the best candidate (**A**) for addition to the cache, 640. In one embodiment, **A** is the SA with the highest SA metric value. For example, an SA metric associated with **A** may be increased by a driver agent if **A** is added to the driver agent, and the traffic stream may be highly active, resulting in a high SA metric value.

[0034] The SA metric values of **A** and **B** are compared, 650. In one embodiment, **A** will only be added to the cache if the metric value of **A** is greater than **B** by at least **C**. In one embodiment, **C** is a predetermined value based on experimentation that estimates the cost of cache entry replacement, for example, 100. For example, **C** may be a high number if cache replacement results in large delays or packet loss, or **C** may be zero if there is little or no cost associated with cache replacement. If the value of the SA metric for **A** is not greater than the SA metric for **B** by at least **C**, no cache entries will be replaced, 690. If the SA metric for **A** is greater than the SA metric for **B** by at least **C**, then **B** is removed from the cache, 660.

[0035] **A** is added to the cache and the value **X** is added to the SA metric for **A**, 670. In one embodiment, **X** is a predetermined value based on experimentation, for example, 10,000. Adding **X** to the SA metric for **A** gives advantage to **A** so that it is likely remain in the cache. Timer **T1** is reset, 680.

[0036] **Figure 7** is one embodiment of a flow diagram for decreasing the value of the metrics of the SAs. A timer **T2** expires indicating a timeout condition, 710. A timer can be any form of counter, timer, capture and compare, or other device or method used to indicate to a computer system process when a time-dependent operation or series of operations should be performed.

Detection of the timeout may be performed by either software or hardware interrupt, or by polling. The timeout period for **T2** is determined by experimentation, for example, one second.

[0037] In one embodiment, the expiration of timer **T2** indicates to a system that a predetermined policy for decreasing the SA metric value of all SAs should be performed. Decreasing the SA metrics may be performed by subtracting a value from the SA metrics, by shifting the SA metric value to the right, or by dividing down the SA metrics. For example, a system processor can have a timer that causes an interrupt on overflow to indicate when to execute a command to a driver agent to perform a numerical operation to decrease the value of the SA metrics.

[0038] The value of the SA metric of all SAs is decreased, 720. For example, the driver agent can divide the metric values by two. In one embodiment, decreasing the SA metric value ages the metrics to give advantage to an SA that has more recent activity over a cached SA associated with a traffic stream that previously had heavy traffic that subsequently fell into disuse. Timer **T2** is reset, 730.

[0039] In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes can be made thereto without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.